Enhanced Pattern Matching Performance Using Improved Boyer Moore Horspool Algorithm

Dai Hong
College of Software, Liaoning University of Science & Technology,
Anshan, Liaoning, 114051, China
E-mail: dear_red9@163.com

Abstract

In computer science, the Boyer–Moore–Horspool algorithm is an algorithm for finding substrings in strings. A pattern matching problem can be classified into software and hardware based on implemental methods. It is important to enhance pattern matching performance. This paper proposes enhanced pattern matching performance using improved Boyer Moore Horspool Algorithm. It combines the deterministic finite state; the improved Boyer Moore Horspool Algorithm takes full use of the matching information to skip several characters. Experimental result shows that the proposed improved Boyer Moore Horspool Algorithm is effective.

Keywords: Pattern Matching, Boyer Moore Horspool algorithm, Shift Function

1. Introduction

The examples of performing pattern matching in the string can be used in text editing applications, intrusion detection system, kill virus, package filter firewall system. String matching plays a fundamental role in many network security applications such as NIDS, virus detection and information filtering. Jianlong Tan et al. [1] proposed cache-efficient methods to accelerate classical multiple string matching algorithms. They observed that most classical algorithms perform poorly as pattern set grows due to their high memory requirement and the poor cache behavior. Based on this observation, they proposed efficient methods employing cache-efficient strategies, i.e., to accelerate string matching by minimizing memory usage and maximizing cache locality.

A pattern matching problem can be classified into software and hardware based on implemental methods. For software matching algorithms, based on the number of patterns to be matched at a time, a pattern matching problem can be classified into single pattern and multiple pattern matching problems. The difference between them is that the single pattern matching algorithm searches through the text to match only one given pattern at one scan, while the multiple patterns matching algorithm searches through the text to match a given set of patterns at one scan. There are three primary single pattern matching algorithms: Knuth-Morris-Pratt’s algorithm, Boyer-Moore’s algorithm, and Karp-Rabin’s algorithm. According to definition mode of successful matching, a pattern matching problem can be classified into precise matching and blur matching; According to design thoughts of matching algorithms, it can be classified into three types. To solve the multiple-pattern matching problem, different algorithms are proposed based on the finite state automata (FSA) which implemented by either software or hardware.

S. Chidambaram et al. [2] investigated the use of Galois LFSRs as test pattern generators in BIST schemes that employ multiple scan chains. Current schemes use LFSRs or cellular automata (CA) with additional phase shifters to provide guaranteed minimum phase shifts between successive scan chains and also impose an upper bound on the number of taps for the XOR gate of each phase shifter. They compared CA with phase shifters in terms of the minimum inter-channel separation that they achieve and the overall XOR cost for each construction.

Wanli Ouyang et al. [3] proposed an analysis and comparison of state-of-the-art algorithms for full search equivalent pattern matching. Our intention is that the datasets and tests used in our evaluation will be a benchmark for testing future pattern matching algorithms, and that the analysis concerning the state-of-the-art algorithms could inspire new fast algorithms. They also proposed extensions of the evaluated algorithms and show that they outperform the original algorithms.

Based on the discussion and comparison of BMH, BMHS string matching algorithm, The optimum...
choice of which BMHS algorithm is improved, BMHS algorithm for the shortcomings, that is the algorithm fails in matching text strings match's last bit characters to participate in the next match, in the case of a series of characters appearing, It can't achieve a maximum moving distance \( m+1 \). Linquan Xie et al. [4] proposed an improved algorithm of BMHS2, in most cases, the improved algorithm of BMHS2 which can achieve maximum moving distance \( m+1 \).

Nikzad Babaii Rizvandi et al. [5] studied CPU utilization time patterns of several applications. After extracting running patterns of several applications, they are saved in a reference database to be later used to system parameters to efficiently execute unknown applications in future. To achieve this goal, CPU utilization patterns of new applications are compared with the already known ones in the reference database to find their most probable execution patterns.

With the development of high-speed network technique and increasing volume of network traffic, traditional pattern matching method can’t adapt to the new challenges to intrusion detection. To solve this, protocol analysis is introduced into the procedure of intrusion detection, and it has advantages such as the capability of detailed command parsing, attack detection and protocol acknowledgement against fragment attacks, the lower false positives and high performance. By the integration with pattern matching, intrusion detection technology based on protocol analysis may significantly reduce the amount of computation and improve the efficiency of packet analysis as well as the detection rates. [6]

Durga Toshniwal et al. [7] proposed an algorithm for clustering unstructured text documents using Naive Bayesian concept and shape pattern matching. The Vector Space Model is used to represent our dataset as a term-weight matrix. In any natural language, semantically linked terms tend to co-occur in documents. Hence, the co-occurrences of pairs of terms in the term-weight matrix are observed.

Benny Porat et al. [8] presented a fully online randomized algorithm for the classical pattern matching problem that uses merely space, breaking the \( O(m) \) barrier that held for this problem for a long time. Their method can be used as a tool in many practical applications, including monitoring Internet traffic and firewall applications. After the preprocessing phase, the characters of the text \( T \) of size \( n \) arrive one at a time in an online fashion. For each index of the text input they indicate whether the pattern matches the text at that location index or not.

This paper proposes enhanced pattern matching performance using improved Boyer Moore Horspool algorithm. It is an efficient pattern matching algorithm. It combines the deterministic finite state; the improved Boyer Moore Horspool Algorithm takes full use of the matching information to skip several characters. It especially fit the case which lots of characters sets.

The rest of the paper is organized as follows. In section 2, we introduce improved Boyer Moore Horspool algorithm. Section 3 focuses on experiments and evaluations. Finally, we conclude the paper with some remarks.

2. Improved Boyer Moore Horspool algorithm

The paper puts forward a new algorithm that it modifies Boyer-Moore-Horspool algorithm, it calls MBMH for short, and it is different from BMH in three aspects.

1) When mismatching happens and pattern strings aligns substrings of text string \( T'_{i\leftarrow j} \) according to \( t_{i\leftarrow m} \) and \( t_{i\leftarrow m+1} \) ascertain shift distance.

2) Need build two shift tables, set up \( \text{shift} \) and \( \text{shift0} \). Building processing of shift table is the same as shift table of BMH algorithm. According to \( t_{i\leftarrow m+1} \) appear the first position from right to left in the substrings of pattern strings \( \{p_0p_1p_2...p_{m-1}\} \) ascertain shift value. According as \( t_{i\leftarrow m} \) appear the first position from right to left in the substrings of pattern strings \( \{p_0p_1p_2...p_{m-1}\} \) ascertain \( \text{shift0} \) value. If occur, then shift pattern strings to right, make \( t_{i\leftarrow m} \) align the first same character of pattern strings. If disappear, then shift pattern strings to right \( m+1 \) characters distances. The definition as follows:

\[
\text{shift0(char)} = \begin{cases} 
  m+1; & \text{char} \neq \text{pattern}[j], 0 \leq j \leq m - 1, \\
  \text{character char is not exist in pattern} & \\
  j = \max\{|i, \text{pattern}[j]=\text{char}, 0 \leq j \leq m - 1\} & \\
  \text{others} & 
\end{cases}
\]
For example, pattern="ABCDAAACHA" , the lengths is 9, shift0(B)=8 , shift0 (C)=7 , shift0 (D)=6 , shift0 (H)=2 , shift0 (A)=1 . shift0 value of other characters is 10. As follows:

<table>
<thead>
<tr>
<th>Shift0 value</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>A</th>
<th>A</th>
<th>C</th>
<th>H</th>
<th>Other characters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

Shift final distances of pattern strings is maximal value between shift0 (t1+m) and shift0 (t1+m-1). So, increase more shift distances than way of making simply use of t1+m-1.

The paper proposes new multiple patterns matching algorithm, Improved Boyer-Moore-Horspool algorithm (we call it IBMH for short), it combines deterministic finite state automata (DFSA) with MBMH algorithm. Not only realizes multiple patterns precision matching, but also immediately shift by using bad character heuristic. Compared with AC algorithm, not only saves memory space, but also increases matching speed, especially when pattern string character sets are less than text string character sets.

Processing of pretreatment pattern strings includes two aspects: (1) Utilize all pattern strings to construct skip and skip0 functions. (2) DFSA is initially constructed from a given set of pattern, include state transition goto function and output function. Current state and input character from text string continues mapping another state by state transition function. If a final state is reached, the patterns specified by the output function are matched. During the state transition process, the goto function maps initial state 0, when an input character causes a mismatching. When it occurs, the skip and skip0 function are determined to shift leftward state tree distance, and the state transition restarts from state 0. Matching process of pattern strings restarts state transition process from state 0.

Text strings are confirmed as DFSA of inputs. They perform multiple pattern strings matching at the same time. Pattern strings are constructed a state tree by IBMH algorithm in the constructed state transition goto function process. In the beginning, according to minimum length pattern string, state tree is aligned text strings right. State tree is aligned substring \( t_{n-minlen}^{-1} t_{n-minlen+1}^{-1} t_{n-1} \) of text strings left. Initial state 0 is current input state of state automata. The first character \( t_{n-minlen} \) of substring of text strings is aligned state tree which is current input character of state automata.

If \( goto(0, t_{n-minlen}) = S \), \( S \neq 0 \), then state automata transfers state, \( S \) is current state, the next character of text strings \( t_{n-minlen+1} \) is current input character of state automata. Visiting characters in the text strings from left to right. In addition, if output( \( S \) ) \( \neq empty \), then output( \( S \) ) appointed pattern matching strings is successful.

Continue estimate \( goto(s, t_{n-minlen+1}) \), until state \( s' \) mismatching character \( a \) in the substrings of text strings, make \( goto(s', a) = 0 \). At the moment, we need not consider mismatching characters. Only take into account left character \( t_{i+m-1} \) and the first character of substring of text strings \( t_{n-minlen} \), which is aligned state tree. According to the maximal value between skip \( t_{n-minlen} \) and skip0 \( t_{n-minlen-1} \) confirms slide distances of state tree leftward. Shift direction of state tree is from right to left.

By discussing constructed algorithm of state transition function of AC algorithm, we sum up state transition goto function and output function in the new algorithm. State transition function of AC algorithm makes a state and an input character map a state or failure message. State transition goto function mapping the all failure message instead pointed state in the algorithm.

If pattern string set is \{she, her, sher, hsri\}, obtains state transition figure and output function. Numbers with circle in the figure 1 express final state. For each state, besides indicative characters on the line, state transfer from a current state to an initial state 0. In order to simple state transition figure, output transition is not express from each state to initial state.
Figure 1. goto figure of pattern string set { she, her, sher, hsri }

Constructed skip and skip0 functions are similar as shift function of MBMH algorithm; it considers characters of all pattern strings in the pattern string sets. In addition, MBMH is a pattern string aligned text string left end. All pattern strings of IBMH algorithm construct state trees. At first, minimum pattern string is aligned text strings right end. So, direction of computing skip and skip0 functions reverse direction of computing shift and shift0 of MBMH.

Skip function is created according to the first character substring of text strings aligned state tree in all pattern strings except the first character of substring from left to right where the first appear or disappear. Skip0 function is created according to the left character substring of text strings aligned state tree in all pattern strings where the first appear or disappear from left to right. Definition of skip function:

\[
\text{skip}(\text{char}) = \begin{cases} 
\minlen \; ; \\
\text{char} \neq \text{pattern}[]], 1 \leq j \leq m_i - 1 \\
j = \min \{ j \mid \text{pattern}[[j]] = \text{char} \} 
\end{cases}
\]

Minlen is minimum length of all pattern strings, \( m_i \) is length of the \( i \) pattern string. \( k \) is total of pattern strings. The difference of constructing skip and skip0 functions is in that try to search character \( \text{char} \) position right end in pattern strings when computes \( \text{shift}(\text{char}) \). According to the first character aligned in all pattern strings, search the position character \( \text{char} \) position left end in pattern strings when computes \( \text{skip}(\text{char}) \). The computing steps as follows:

a) For every character in ASCII character table, set up \( \text{skip}(\text{char}) = \minlen \).

b) For the \( j+1 \) character in the \( i \) pattern string, set up \( \text{skip}([j]) = \min \{ \text{skip}([j]), \\
(1 \leq j \leq m_i - 1) \}
\]

For example, pattern string character set \{she, hers, his\} is given and sets up \( \text{skip}(e) = 3 \). And then, scans every character of all pattern strings. When scans the third character “e” in the first pattern string, \( \text{skip}(e) \) sets up \( \min[3,2]=2 \). When scans the second character “e” in the second pattern string, \( \text{skip}(e) \) sets up \( \min[2,1]=1 \).

Definition of skip0 function:

\[
\text{skip0}(\text{char}) = \begin{cases} 
\minlen + 1 \; ; \; \text{char} \neq \text{pattern}[], 0 \leq j \leq m_i - 1 \\
d + 1 \; ; \; d = \min \{ d \mid \text{pattern}[[d]] = \text{char} \} \\
(0 \leq d \leq \min len - 1, 1 \leq i \leq k) 
\end{cases}
\]

Skip and skip0 functions of pattern string set \{ she, her, sher, hsri \} is showed in table 1 as follows.
Table 1. Skip function \textit{skip} and \textit{skip0}

<table>
<thead>
<tr>
<th>x</th>
<th>sk</th>
<th>ski</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ip(x)</td>
<td>p0(x)</td>
</tr>
<tr>
<td>s</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>h</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>r</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>i</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>o</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>theers</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. Experiments and evaluations

We will test the shift times of new algorithm. Experiment environment is XP system, Inter Pentium Dual-Core 1.73GHz, memory is 768M, experiment data are notepad file include 256 uppercases, lowercases and numeric random generated in the ASCII table, approximately 1M. In order to stabilize experiment results, taking out strings containing 2, 4, 6, 8, 10, 12, 14 characters from the file endpoint as pattern strings. Run program and get hold of shift time figure of the experiment.

![Figure 2. Shift times in 100,000 characters of text string matching](image)

We can see shift times of new algorithm are less than BMH algorithm from figure 2. So, the matching speed of new algorithm is more rapid than BMH algorithm.

We will test the performance of IBMH algorithm. Experiment environment is XP system, Inter Pentium Dual-Core 1.73GHz, memory is 768M, experiment data are notepad file include 256 uppercases, lowercases and numeric random generated in the ASCII table, approximately 1M. Then strings made of number length among 6 to 10 are randomly generated. The number of strings are 500, 1000, 2000, …, until 10000 in pattern string set. We compared the two algorithms in different situations, and the experimental result is shown in Table 2.
Table 2. Two algorithms visiting characters of text string in experiment

<table>
<thead>
<tr>
<th>Amount of pattern strings</th>
<th>IBMH algorithm</th>
<th>AC algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>65532</td>
<td>86451</td>
</tr>
<tr>
<td>1000</td>
<td>106079</td>
<td>124640</td>
</tr>
<tr>
<td>2000</td>
<td>146537</td>
<td>173604</td>
</tr>
<tr>
<td>3000</td>
<td>156658</td>
<td>213488</td>
</tr>
<tr>
<td>4000</td>
<td>193579</td>
<td>296732</td>
</tr>
<tr>
<td>5000</td>
<td>218653</td>
<td>365217</td>
</tr>
<tr>
<td>6000</td>
<td>239074</td>
<td>547943</td>
</tr>
<tr>
<td>7000</td>
<td>260481</td>
<td>673894</td>
</tr>
<tr>
<td>8000</td>
<td>293498</td>
<td>792451</td>
</tr>
<tr>
<td>9000</td>
<td>324936</td>
<td>857254</td>
</tr>
<tr>
<td>10000</td>
<td>341807</td>
<td>985903</td>
</tr>
</tbody>
</table>

We will analyze difference between IBMH algorithm and AC algorithm.

1) The process of state transition goto function using all pattern strings constructing state tree is similar of AC algorithm. But all failure messages of state transition goto function are pointed to state 0. For random state $S$ and character $a$, so long as $goto(S, a) = 0$, confirm mismatch in compare process. Then slide state tree leftward according to condition. State transition $g$ function makes a state and an input character map a state or failure message in AC algorithm.

Output function in the proposed algorithm is also built in process of state transition goto function. But it doesn’t need fail function when a state mapping string in AC algorithm.

2) Matching from the $n - \text{min len} + 1$ character of text strings and comparing from all pattern strings leftward character. $\text{min len}$ is the minimum length of all pattern strings, $n$ is length of text strings. If the character doesn’t cause a state transition to initial state 0, continue to match $(n - \text{min len})$th characters. Visit text strings from left to right until meet a character which makes the state transfer state 0.

3) When character causes state transition to initial state 0, indicates occur to mismatch. According to $\text{skip}$ and $\text{skip0}$ functions, we judge next compared character position of text string. Register the first visited character position of text strings before shift, need increase a pointer. Character skip value of indicative pointer and a front character $\text{skip0}$ value confirm finial shift distance. Pointer shifts from right to left.

4) The process of building shift $\text{skip}$ and $\text{skip0}$ function is similar as the process of building $\text{shift}$ and $\text{shift0}$ in MBMH algorithm. Only take into account all characters of pattern string sets. According to MBMH method determines shift distances.

Compare multiple patterns matching algorithm with AC algorithm, achieve all pattern strings in file through experiment. Amount characters of two algorithms visit file of table 2. From the table 2, we can see, the IBMH algorithm of multiple patterns matching visits text strings. Range of visiting characters number of text strings increases very small as the number of pattern string number increase, the number is far less than visit characters number of AC algorithm.

The proposed algorithm gets good experiment results because pattern string character sets are less than visit characters number of AC algorithm. Probability that $\text{min len}$ and $\text{min len} + 1$ of shift distance is very far. Therefore, number of visiting text string is very little.

The result shows that the proposed algorithm operates at a much higher efficiency than the other method for matching multiple patterns. From table 2, the proposed algorithm IBMH is efficiency for lots of pattern strings. Range of visiting character number of text strings is very small as pattern strings
increase. Furthermore, when length of pattern string is fixed, number of visiting text string is far less than AC algorithm.

4. Conclusions and future work

It is important of enhance pattern matching performance. This paper proposes enhanced pattern matching performance using improved Boyer Moore Horspool Algorithm. Moreover, it combines the deterministic finite state; the improved Boyer Moore Horspool Algorithm takes full use of the matching information to skip several characters. The proposed algorithm saves more memory resource, especially adapts to pattern string character sets and text character sets, matching speed increases greatly and pattern length and number influence hardly. Experimental result shows that the proposed improved Boyer Moore Horspool Algorithm is effective.

In the common condition, instance experiment validates the case that multiple pattern strings are faster 4 to 5 times than AC algorithm. The number of characters to be inspected decreases as the number of patterns increases. Also, there is a lot of room for improvement, including speed up the proposed algorithm.

5. Acknowledgment

This work is supported by Educational Commission of Liaoning Province of China grant under contract number 2009B107, and also supported by AnShan Science and Technology Development Funds under contract number 2009SF13.

We would like to thank our colleagues and friends for their encouragement and moral support given to us. We would also like to extend our thanks to our family members for all the help, direct or indirect.

6. References